Enhancing ebXML Registries to Make them OWL Aware *

ASUMAN DOGAC asuman@srdc.metu.edu.tr
YILDIRAY KABAK yildiray@srdc.metu.edu.tr
GOKCE B. LALECI banu@srdc.metu.edu.tr
Software Research and Development Center
Department of Computer Engineering
Middle East Technical University (METU)
06531, Ankara, Turkiye

CARL MATTOCKS carlmattocks@checkmi.com
CHECKMi, USA

FARRUKH NAJMI farrukh.najmi@sun.com
Sun Micro Systems, USA

JEFF POLLOCK jeff.pollock@networkinference.com
Network Inference, USA

Abstract. In this paper, we address how ebXML registry semantics support can be further enhanced to make it OWL aware. There are basically three ways of achieving this: The first one is mapping OWL constructs to ebXML registry information model constructs without modifying the registry architecture and implementation. In this way, the semantic explicitly stored in the registry can be retrieved through querying; yet, the application program must contain additional code to process this semantics. The second approach is additionally providing predefined stored procedures in the registry for processing the OWL constructs. We believe that this approach is quite powerful to associate semantics with registry objects: it becomes possible to retrieve knowledge through queries and the enhancements to the registry are generic. The capabilities provided move the semantics support beyond what is currently available in ebXML registries and it does so by using a standard ontology language. The third approach is changing the ebXML registry to support OWL with full reasoning capabilities. However, this approach requires considerable changes in the registry architecture.

Since our aim is to make the ebXML registry OWL aware by keeping the registry specification intact, we take the second approach. To be able to demonstrate the benefits of the enhancements, we also show how the resulting semantics can be made use of in Web service discovery and composition.

This work is realized within the scope of IST-2104 SATINE project as a proposal to OASIS ebXML Semantic Content Management subcommittee which is working on possible semantic extensions to the registry.

* Submitted to the Distributed and Parallel Databases Journal, Kluwer Academic Publishers. This work is supported in part by the European Commission, Project No: IST-1-002104-STP SATINE and by the Scientific and Technical Research Council of Turkey (TÜBİTAK), Project No: EEEAG 104E013

1. Introduction

Currently, semantics is becoming a much broader issue than it used to be since several application domains are making use of ontologies to add the knowledge dimension to their data and applications. One of the driving forces for ontologies is the Semantic Web initiative [2]. As a part of this initiative, W3C's Web Ontology Working Group defined Web Ontology Language (OWL) [31]. OWL is defined as three different sublanguages: *OWL Full*, *OWL DL* and *OWL Lite*, each geared towards fulfilling different requirements [22].

In this paper, we investigate how ebXML registries can be made OWL aware. In this way, we believe a mutually beneficial relationship is created between ebXML registries and the OWL ontology language: the former gains the ability to store OWL ontologies and the mechanisms provided by the ebXML registry proves very useful in associating the semantics defined in an OWL ontology with the registry objects. In this work, we use OWL Lite since we are using ontologies to get knowledge through querying rather than reasoning.

There are three alternatives to support OWL Lite ontologies through ebXML registries:

- Various constructs of OWL can be represented by ebXML classification hierarchies with no changes in the registry architecture specification and implementation. In this way, although some of the OWL semantics stored in an ebXML registry can be retrieved from the registry through ebXML query facilities, further processing needs to be done by the application program to make use of the enhanced semantics. For example, we can introduce "subClassOf" "association" to the ebXML registry to handle OWL multiple inheritance. Yet since ebXML registry does not natively support such an association type, to make any use of this semantics, the application program must have the necessary code, say, to find out all the super classes of a given class.

- The code to process the OWL semantics can be stored in ebXML registry architecture through predefined procedures. For example, to find the super classes of a given class (defined through a new association type of "subClassOf"), a stored procedure can be defined. The user can call this procedure when the need arises. Furthermore, the stored procedures can also be called transparently to the user by changing only the query manager component of the registry.

- The third approach is changing the ebXML registry architecture to support OWL with full reasoning capabilities. Reasoning entails the derivation of new data that is not directly stored in the registry. To deduce this data, rules need to be stored in the registry. However, this approach requires considerable changes in the registry architecture and brings about the efficiency considerations of rule based systems. Since our aim is to make ebXML registry OWL aware rather than specifying a new registry architecture, this approach will not be pursued any further in this paper.

RDF Schema Features	**(In)Equality**	**Property Characteristics**
– *Class (Thing, Nothing)*	– *equivalentClass*	– *ObjectProperty*
– *rdfs: subClassOf*	– *equivalentProperty*	– *DatatypeProperty*
– *rdf:Property*	– *sameAs*	– *inverseOf*
– *rdfs: subPropertyOf*	– *differentFrom*	– *TransitiveProperty*
– *rdfs:domain*	– *AllDifferent*	– *SymmetricProperty*
– *rdfs:range*	– *distinctMembers*	– *FunctionalProperty*
– *individual*		– *InverseFunctionalProperty*
Property Restrictions	**Restricted Cardinality**	**Datatypes**
– *Restriction*	– *minCardinality*	– *xsd datatypes*
– *onProperty*	– *maxCardinality*	**Class Intersection**
– *allValuesFrom*	– *cardinality*	– *intersectionOf*
– *someValuesFrom*		

Figure 1. OWL Lite Constructs

Being OWL aware entails the following:

- Representing OWL constructs through ebXML constructs.
- Automatically generating ebXML constructs from the OWL descriptions and storing the resulting constructs into the ebXML registry.
- Querying the registry for enhanced semantics.

Furthermore, we show how the resulting semantics can be made use of in Web service discovery and composition. This work is realized within the scope of IST-2104 SATINE project as a proposal to OASIS ebXML Semantic Content Management subcommittee which is working on possible semantic extensions to the registry.

The paper is organized as follows: Section 2 briefly summarizes the main technologies involved in this work, namely, OWL and ebXML Registry architecture. In Section 3, we give an overall view of the approach and describe how the proposed enhancements fit into ebXML architecture. Section 4 describes how semantics defined in OWL ontologies can be represented and accessed in ebXML registries. Section 5 put the work described in this paper into perspective by summarizing the ebXML semantic standardization efforts undertaken by the OASIS open source standards body. Section 6 gives the related work. Finally, Section 7 concludes the paper and presents the future work.

2. OWL and ebXML RIM

In order to describe how OWL ontologies can be stored in ebXML registries we first briefly summarize the semantic constructs they each provide.

2.1. Web Ontology Language (OWL)

Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the World Wide Web [31]. OWL is derived from the DAML+OIL Web Ontology Language [7] and builds upon the Resource Description Framework (RDF) [37, 38].

OWL describes the *structure* of a domain in terms of *classes* and *properties*. Classes can be names (URIs) or *expressions* and the following set of *constructors* are provided for building class expressions: *owl:intersectionOf, owl:unionOf, owl:complementOf, owl:oneOf, owl:allValuesFrom, owl:someValuesFrom*, and *owl:hasValue*.

In OWL, properties can have multiple domains and multiple ranges. Multiple domain (range) expressions restrict the domain (range) of a property to the intersection of the class expressions.

Another aspect of the language is the axioms supported. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties [19]. The following are the set of OWL axioms: *rdfs:subClassOf, owl:sameClassAs, rdfs:subPropertyOf, owl:samePropertyAs, owl:disjointWith, owl:sameIndividualAs, owl:differentIndividualFrom, owl:inverseOf, owl:transitiveProperty, owl:functionalProperty*, and *owl:inverseFunctionalProperty*. OWL constructs are given in more detail in Section 4.1 while describing their representation in ebXML registry.

OWL provides three decreasingly expressive sublanguages [41]:

- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. It is unlikely that any reasoning software will be able to support complete reasoning for *OWL Full* [22].
- *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). *OWL DL* is so named due to its correspondence with description logics which form the formal foundation of OWL.
- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints.

Within the scope of this paper, we consider *OWL Lite* constructs which are given in Figure 1 and in the rest of the paper; OWL is used to mean *OWL Lite* unless otherwise stated.

2.2. ebXML Registry Architecture and Information Model

An ebXML registry consists of both a registry and a repository. The repository is capable of storing any type of electronic content, while the registry is capable

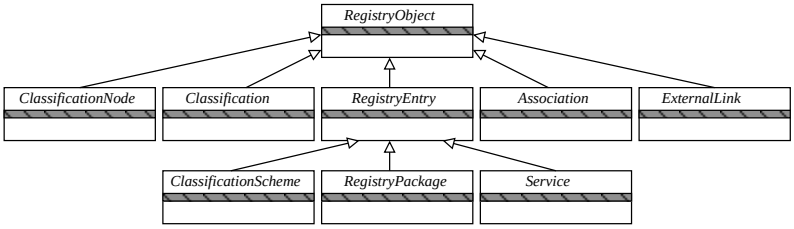

Figure 2. A Part of the ebXML RIM Class Hierarchy

of storing metadata that describes content. The content within the repository is referred to as "repository items" while the metadata within the registry is referred to as "registry objects". Clients access the registry and the repository via the ebXML registry API as defined in [16]. The API has two main interfaces:

- LifeCycleManager (LCM) is the interface responsible for all object lifecycle management requests.
- QueryManager (QM) is the interface responsible for handling all query requests.

The LifeCycleManager service enforces the life cycle rules for objects. The QueryManager interface of the ebXML Registry API provides access to the query service of the ebXML registry. A client uses the operations defined by this service to query the registry and discover objects. Supported query syntaxes include:

- An XML Filter Query syntax,
- An SQL-92 query, and
- A stored query syntax that allows client to invoke queries stored in the server by simply identifying the parameterized query and providing parameters for the query.

2.2.1. ebXML Registry Information Model

The ebXML registry defines a Registry Information Model (RIM) [15] which specifies the standard metadata that may be submitted to the registry. This complements the ebXML Registry API which defines the interface clients may use to interact with the registry. Figure 2 presents the part of the ebXML RIM [15] related with storing metadata information. The main features of the information model include:

- RegistryObject: The top level class in RIM is the "RegistryObject". This is an abstract base class used by most classes in the model. It provides minimal metadata for registry objects.

- Object Identification: All RegistryObjects have a globally unique id, a human friendly name and a human friendly description.
- Slot: "Slot" instances provide a dynamic way to add arbitrary attributes to "RegistryObject" instances.
- Object Classification: Any RegistryObject may be classified using ClassificationSchemes and ClassificationNodes which represent individual class hierarchy elements. A ClassificationScheme defines a tree structure made up of "ClassificationNode"s. The ClassificationSchemes may be user-defined.
- Object Association: Any RegistryObject may be associated with any other RegistryObject using an Association instance where one object is the sourceObject and the other is the targetObject of the Association instance. An Association instance may have an associationType which defines the nature of the association. There are a number of predefined *Association Types* that a registry must support to be ebXML compliant [15] as shown in Table 1. ebXML allows this list to be expanded.
- Object Organization: RegistryObjects may be organized in a hierarchical structure using a familiar file and folder metaphor. The RegistryPackage instances serve as folders while RegistryObjects server as files in this metaphor. In other words RegistryPackage instances group logically related RegistryObject instances together.
- Service Description: The Service, ServiceBinding and SpecificationLink classes provide the ability to define service descriptions including WSDL and ebXML CPP/A.

As a summary, ebXML registry provides a persistent store for registry content. The current registry implementations store registry data in a relational database. ebXML Registry Services Specification defines a set of Registry Service interfaces which provide access to registry content. There are a set of methods that must be supported by each interface. A registry client program utilizes the services of the registry by invoking methods on one of these interfaces. The Query Manager component also uses these methods to construct the objects by obtaining the required data from the relational database through SQL queries. In other words, when a client submits a request to the registry, registry objects are constructed by retrieving the related information from the database through SQL queries and are served to the user through the methods of these objects.

3. Proposed Enhancements to the ebXML Registry Architecture

Being OWL aware entails the following enhancements to the ebXML registry:

- *Representing OWL constructs through ebXML constructs*: ebXML provides a classification hierarchy made up of classification nodes and predefined type of

Name	Description
RelatedTo	Defines that source RegistryObject is related to target RegistryObject.
HasMember	Defines that the source RegistryPackage object has the target RegistryObject object as a member.
ExternallyLinks	Defines that the source ExternalLink object externally links the target RegistryObject object.
Contains	Defines that source RegistryObject contains the target RegistryObject.
EquivalentTo	Defines that source RegistryObject is equivalent to the target RegistryObject.
Extends	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
Implements	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
InstanceOf	Defines that source RegistryObject is an Instance of target RegistryObject.
Supersedes	Defines that the source RegistryObject supersedes the target RegistryObject.
Uses	Defines that the source RegistryObject uses the target RegistryObject in some manner.
Replaces	Defines that the source RegistryObject replaces the target RegistryObject in some manner.
SubmitterOf	Defines that the source Organization is the submitter of the target RegistryObject.
ResponsibleFor	Defines that the source Organization is responsible for the ongoing maintainence of the target RegistryObject.
OffersService	Defines that the source Organization object offers the target Service object as a service.

Table 1. Predefined Association Types in ebXML Registries

associations between the registry objects. We represent *OWL Lite* constructs by using combinations of these constructs and define additional types of associations when necessary. For example, "OWL ObjectProperty" is defined by introducing a new association of type "objectProperty". The details of this work are presented in Section 4.

- *Automatically generating ebXML constructs from the OWL descriptions and storing the resulting constructs into the ebXML registry*: We developed a tool to create an ebXML Classification Hierarchy from a given OWL ontology automatically by using the transformations described in Section 4. The OWL file is parsed using Jena [21], the classes together with their property and restrictions

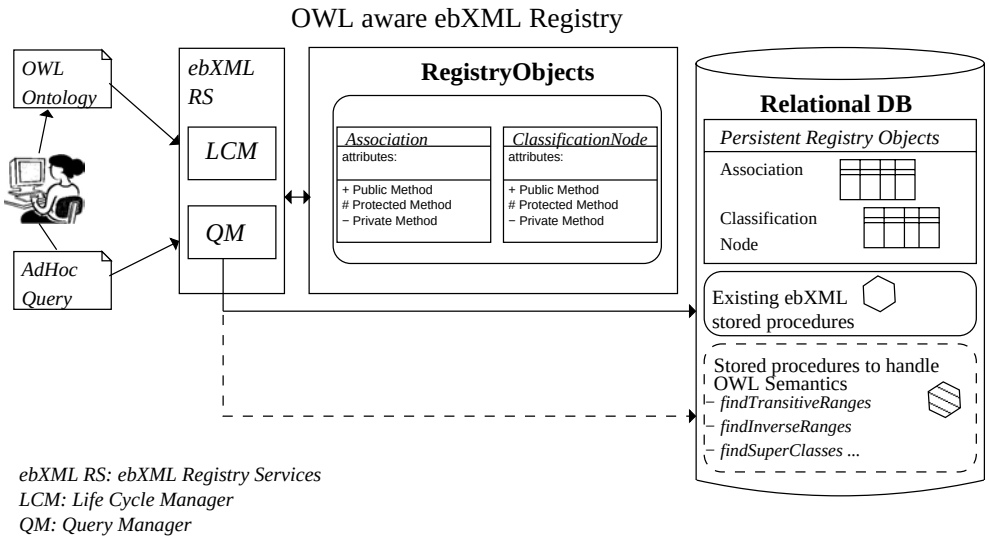

Figure 3. Enhancements to the ebXML Registry Architecture

are identified, and the "SubmitObjectsRequest" is prepared automatically. This request is then sent to ebXML registry which in turn creates necessary classes and associations between them.

- Querying the registry for enhanced semantics: We provide additional stored procedures to process the OWL semantics introduced in Section 4. A user can handle the OWL semantics by using these stored procedures or through SQL. Note that stored procedures and SQL are two of the supported query syntaxes in ebXML. In order to handle the OWL semantics transparently to the user through the third query syntax, namely, the "filter query", the Query Manager needs to be modified to invoke the related stored procedures we have introduced, when necessary.

The enhanced architecture is shown in Figure 3. The OWL constructs are represented entirely through ebXML constructs by defining new types of associations which is allowed by the registry architecture. Hence there are no changes in the relational database schemas.

4. Providing OWL Lite Support to ebXML Registries

In this section, we first describe how OWL constructs can be represented through ebXML registry information model constructs. We then provide the stored pro-

Table 2. ebXML Relational Schemas

ClassificationNode(accessControlPolicy, id, objectType, code, parent, path)
Association(accessControlPolicy, id, objectType, associationType, sourceObject, targetObject, isConfirmedBySourceOwner, isConfirmedByTargetOwner)
Name_(charset, lang, value, parent)

cedures to retrieve richer sets of results from the registry based on *OWL Lite* constructs. The stored procedures are defined using the ebXML relational schema specifications. The schemas used in the examples are given in Table 2.

In Section 4.2, to demonstrate the benefits of the additional semantics incorporated into the ebXML registries, we describe a semantic-based service composition tool. This tool partially automates service discovery and composition in OWL aware ebXML registries.

4.1. Mapping OWL Ontologies through ebXML Classification Hierarchies and Providing Registry Support for Processing the OWL Constructs

From the descriptions presented in Section 2, it is clear that there are considerable differences between an OWL ontology and an ebXML class hierarchy in terms of semantic constructs. In this section, we provide the details of representing the *OWL Lite* constructs in an ebXML registry and then give the required stored procedures to process this semantics.

4.1.1. OWL Classes and Properties

OWL classes can be represented through "ClassificationNodes" and RDF properties that are used in OWL can be treated as "Associations". An "Association" instance represents an association between a "source RegistryObject" and a "target RegistryObject". Hence the target object of "rdfs:domain" property can be mapped to a "source RegistryObject" and the target object of "rdfs:range" can be mapped to a "target RegistryObject". In OWL, properties can be of two types:

- ObjectProperty type defines relations between instances of two classes.
- DatatypeProperty type defines relations between instances of classes and XML Schema datatypes.

To represent OWL ObjectProperty (or DatatypeProperty) in ebXML, we define a new type of association called "ObjectProperty" (or "DatatypeProperty"). Consider the following example which defines an object property "hasAirport" whose domain is "City" and whose range is "Airport":

```
<owl:ObjectProperty rdf:ID="hasAirport">
   <rdfs:domain rdf:resource="#City"/>
```

```
   <rdfs:range rdf:resource="#AirPort"/>
</owl:ObjectProperty>
```

In order to define this property in ebXML RIM, first, two classification nodes are created, namely "City" and "Airport". Then, an association, called "hasAirport" of type "ObjectProperty", is defined where the "sourceObject" is "City" and the "targetObject" is "Airport", as shown in the following:

```
<rim:ClassificationNode id = 'City'parent= 'Country'> </rim:ClassificationNode>
<rim:ClassificationNode id = 'Airport' parent= 'TravelThing'> </rim:ClassificationNode>
<rim:Association id = 'promotion'associationType = 'ObjectProperty' sourceObject =
'City' targetObject = 'Airport' >
</rim:Association>
```

Similarly, to represent OWL DatatypeProperty in ebXML, we define a new type of association called "DatatypeProperty". Consider the following example which defines an datatype property "hasPrice" whose domain is the "AirReservationServices" and whose range is "XMLSchema nonNegativeInteger":

```
<owl:DatatypeProperty rdf:ID="hasPrice">
  <rdfs:subpropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/05/Profile.owl"/>
  <rdfs:domain rdf:resource="#AirReservationServices"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema/nonNegativeInteger"/>
</owl:DatatypeProperty>
```

To describe this semantics, we define a new association of type "DatatypeProperty" as shown in the following:

```
<rim:Association id = 'hasPrice' associationType = 'DatatypeProperty'
   sourceObject = 'AirReservationServices'
   targetObject = 'integer' >
   <rim:Name> <rim:LocalizedString value ="hasPrice"/></rim:Name>
</rim:Association>
```

OWL allows the use of XML Schema datatypes to describe part of the datatype domain by simply including their URIs within an OWL ontology. In ebXML, XML Schema datatypes are used by providing an external link from the registry, as demonstrated in the following:

```
<rim:ExternalLink id = "integer"
                externalURI="http://www.w3.org/2001/XMLSchema#integer" >
      <rim:Name> <rim:LocalizedString value = "XML Schema integer"/>
                </rim:Name>
</rim:ExternalLink>
```

Once such ObjectProperty or DatatypeProperty definitions are stored in the ebXML registry, they can be retrieved through ebXML query facilities by the user. However, providing some stored procedures for this purpose facilitates the direct access. We therefore propose the following stored procedure to be available in the registry which retrieves all the object properties of a given classification node:

```
CREATE PROCEDURE findObjectProperties($className) AS
BEGIN
SELECT A.id
FROM Association A, Name_ N, ClassificationNode C
WHERE A.associationType LIKE 'objectProperty' AND
      C.id = N.parent AND
      N.value LIKE $className AND
      A.sourceObject = C.id
END;
```

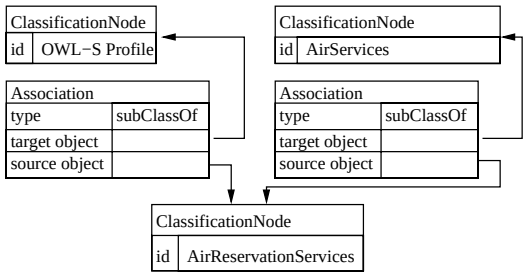

Figure 4. Representing an Example "owl:subClassOf" Property in ebXML Registry

A similar stored procedure can be given to retrieve datatype properties of a given class.

4.1.2. OWL Class Hierarchies

When it comes to mapping OWL class hierarchies to ebXML class hierarchies, OWL relies on RDF Schema for building class hierarchies through the use of "rdfs:subClassOf" property and allows multiple inheritance. An ebXML Class hierarchy has a tree structure, and therefore is not readily available to express multiple inheritance, that is, there is a need for additional mechanisms to express multiple inheritance. We define a "subClassOf" property as an association for this purpose.

Consider the example:

```
<owl:Class rdf:ID="AirReservationServices">
 <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-s/1.0/Profile.owl#Profile"/>
 <rdfs:subClassOf  rdf:resource="#AirServices"/>
</owl:Class>
```

Here, "AirReservationServices" service inherits both from "AirServices" service and OWL-S ServiceProfile class. Figure 4 shows how this is represented through ebXML RIM constructs.

Once we define such a semantics, we need the code to process the objects in the registry according to the semantics implied; that is, given a class, we should be able to retrieve all of its subclasses and/or all of its super classes. By making the required stored procedures available in the registry, this need can be readily served. For example, the following procedure finds all the immediate super classes of a given class:

```
CREATE PROCEDURE findSuperClasses($className) AS
BEGIN
SELECT C2.id
FROM Association A, Name_ N, ClassificationNode C1, ClassificationNode C2
WHERE A.associationType LIKE 'subClassOf' AND
      C1.id = N.parent AND
```

```
      N.value LIKE $className AND
      A.sourceObject = C1.id AND
      A.targetObject = C2.id
END;
```

Similar procedures can be provided to find all the superclasses of a given class (not only the immediate ones) as well as all its subclasses. The following procedure can then be used to retrieve all of the properties of a given class including the ones inherited from its super classes:

```
CREATE PROCEDURE findInheritedObjectProperties ($className) AS
SELECT A.id FROM Association A, ClassificationNode C WHERE
A.sourceObject=C.id AND
      A.associationType LIKE 'objectProperty' AND
      C.id IN (
            SELECT parent
            FROM name_
            WHERE value LIKE $className
            UNION
            findSuperClasses($className)
            }
END;
```

4.1.3. OWL subPropertyOf

Since OWL properties are represented through ebXML associations, we define "rdfs:subPropertyOf" as an association between associations with a new association type of "subPropertyOf". The following procedure finds all the immediate super properties of a given property and similar procedures can be made available for all the super and subproperties:

```
CREATE PROCEDURE findSuperProperties($propertyName) AS
BEGIN
SELECT A3.id
FROM Association A1, Association A2, Association A3, Name_ N
WHERE A2.associationType LIKE 'subPropertyOf' AND
      A1.id = N.parent AND
      N.value LIKE $propertyName AND
      A2.sourceObject = A1.id AND
      A2.targetObject = A3.id
```

4.1.4. OWL equivalentClass, equivalentProperty and sameAs Properties

In ebXML, the predefined "EquivalentTo" association (Table 1) expresses the fact that the source registry object is equivalent to target registry object. Therefore, "EquivalentTo" association is used to express "owl:equivalentClass", "equivalentProperty" and "sameAs" properties since classes, properties and instances are all ebXML registry objects.

Given a class, the following stored procedure retrieves all the equivalent classes:

```
CREATE PROCEDURE findEquivalentInstances($className) BEGIN SELECT
N.value FROM Service S, Name_ N WHERE S.id IN (
    SELECT classifiedObject
```

```
    FROM Classification
    WHERE classificationNode IN (
        SELECT id
        FROM ClassificationNode
        WHERE id IN (
            SELECT parent
            FROM name_
            WHERE value LIKE $className
            )
        UNION
        SELECT A.targetObject
        FROM Association A, Name_ N, ClassificationNode C
        WHERE A.associationType LIKE 'EquivalentTo' AND
              C.id = N.parent AND
              N.value LIKE $className AND
              A.sourceObject = C.id
            )
    ) AND S.id=N.parent
END;
```

4.1.5. OWL Transitive Property

In OWL, if a property, P, is specified as transitive then for any x, y, and z: P(x,y) and P(y,z) implies P(x,z). Transitive property can be defined as a new type of association in ebXML.

Consider the following example where we define the "succeeds" as a transitive property of "TravelWebService" class:

```
<owl:ObjectProperty rdf:ID="succeeds">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="#TravelWebService" />
  <rdfs:range rdf:resource="#TravelWebService" />
</owl:ObjectProperty>
```

Assuming the following two definitions:

```
<TravelWebService rdf:ID="MyHotelAvailabilityService">
  <succeeds rdf:resource="#MyAirReservationService" />
</TravelWebService>

<TravelWebService rdf:ID="MyInsuranceService">
  <succeeds rdf:resource="#MyHotelAvailabilityService" />
</TravelWebService>
```

Since "succeeds" is a transitive property, it follows that "MyInsuranceService" succeeds "MyAirReservationService" although this fact is not explicitly stated.

To make any use of this transitive property in ebXML registries, coding is necesary to find out the related information. We provide the following stored procedure to handle this semantics: Given a class which is a source of a transitive property, this stored procedure retrieves not only the target of a given transitive property, but if the target objects have the same property, it also retrieves their target objects too.

```
CREATE PROCEDURE findTransitiveRelationships($className,
$propertyName) BEGIN SELECT A2.targetObject FROM Association A1,
Association A2, Name_ N1,Name_ N2, Name_ N3 WHERE
A1.associationType LIKE 'transitiveProperty' AND
      A1.id = N1.parent AND
```

```
      N1.value LIKE $propertyName AND
      A1.sourceObject = N3.parent AND
      N3.value LIKE $className AND
      A2.sourceObject = A1.targetObject AND
      A2.id = N2.parent AND
      N2.value LIKE $propertyName AND
      A2.associationType LIKE 'transitiveProperty'
UNION
SELECT A1.targetObject
FROM Association A1, Name_ N1, Name_ N3
WHERE A1.associationType LIKE 'transitiveProperty' AND
      A1.id = N1.parent AND
      N1.value LIKE $propertyName AND
      A1.sourceObject = N3.parent AND
      N3.value LIKE $className
END;
```

4.1.6. OWL inverseOf Property

In OWL, if a property, P1, is tagged as the "owl:inverseOf" P2, then for all x and y: P1(x,y) iff P2(y,x). Consider for example the "succeeds" property defined in Section 4.1.5. To denote that a certain Web service instance precedes another, we may define the "precedes" property as an inverse of the "succeeds" property as follows:

```
<owl:ObjectProperty rdf:ID="precedes">
  <owl:inverseOf rdf:resource="#succeeds" />
</owl:ObjectProperty>
```

Then, by using the following stored procedure, we can find all the services that precede a given service by making use of its "succeeds" property.

```
CREATE PROCEDURE findInverseRanges($className, $propertyName)
BEGIN
SELECT C2.id
FROM Association A, Name_ N, Name_ N2, ClassificationNode C1, ClassificationNode C2
WHERE A.id=N2.parent AND
      N2.value LIKE $propertyName AND
      C1.id = N.parent AND
      N.value LIKE $className AND
      A.sourceObject = C1.id AND
      A.targetObject = C2.id
UNION
SELECT A3.sourceObject
FROM Association A1, Association A2, Association A3, Name_ N, NAME_ N2, ClassificationNode C1
WHERE A2.associationType LIKE 'inverseOf' AND
      A1.id = N.parent AND
      N.value LIKE $propertyName AND
      A2.sourceObject = A1.id AND
      A3.id=A2.targetObject AND
      C1.id = N2.parent AND
      N2.value LIKE $className AND
      A3.targetObject = C1.id
END;
```

4.1.7. OWL Restriction

Another important construct of OWL is "owl:Restriction". In RDF, a property has a global scope, that is, no matter what class the property is applied to, the range of the property is the same. "owl:Restriction", on the other hand, has a local scope; restriction is applied on the property within the scope of the class where it is defined. The aim is to make ontologies more extendable and hence more reusable. OWL provides the following language elements to indicate the type of restriction: *owl:allValuesFrom, owl:someValuesFrom, owl:hasValue*. An *owl:allValuesFrom* element defines the class of all objects for whom the values of property all belong to the class expression.

Consider the following example:

```
<owl:Class rdf:ID="AirReservationServices">
  <rdfs:subClassOf rdf:resource="&service"/>
  <rdfs:subClassOf rdf:resource= "#AirServices"/>
  <rdfs:subClassOf>
    <owlRestriction>
    <owl:onProperty rdf:resource="#paymentMethod"/>
    <owl:allValuesFrom rdf:resource= "#PossiblePaymentMethods"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Here "owl:Restriction" defines an anonymous class, that is the class of all things that satisfy this restriction. The restriction is that the property "paymentMethod" should get all of its values from the class "PossiblePaymentMethods". By defining "AirReservationServices" class as a subclass of this anonymous class, its "paymentMethod" property is restricted to the elements of the "PossiblePaymentMethods".

In ebXML class hierarchies, on the other hand, an association (which represents a property) is already defined in a local scope by associating two nodes of the class hierarchy. The type of the restriction can be expressed by special slot values. Figure 5 shows how the example above is represented through ebXML RIM constructs.

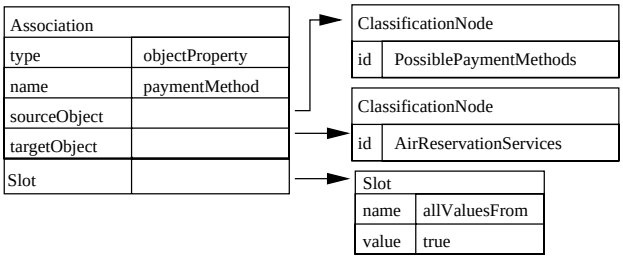

Figure 5. Representing an Example OWL Restriction in ebXML Registry

4.1.8. OWL Class Intersection

OWL provides the means to manipulate class extensions using basic set operators. In *OWL Lite*, only "owl:intersectionOf" is available which defines a class that consists of exactly all objects that do not belong to both of the classes. Consider the following example:

```
<owl:Class rdf:ID="AirReservationServices">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AirServices" />
    <owl:Class rdf:about="#ReservationServices" />
  </owl:intersectionOf>
</owl:Class>
```

In ebXML RIM "owl:intersectionOf" set operator can be expressed as follows:

- A new association type called "intersectionOf" is created.
- The classes constituting the intersection are represented as members of a RegistryPackage.
- The source object of the set operator is assigned as the sourceObject of the "intersectionOf" association.
- The target object of the "intersectionOf" association is set to be the newly created RegistryPackage.

The RIM representation of the OWL example presented above is as follows:

```
<rim:ClassificationNode id = 'AirServices'  parent= 'TravelServices'>
    <rim:Name> <rim:LocalizedString value = 'AirServices' /> </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode id = 'ReservationServices'  parent= 'TravelServices'>
    <rim:Name> <rim:LocalizedString value = 'ReservationServices' /> </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode id = 'AirReservationServices'  parent= 'TravelServices'>
    <rim:Name> <rim:LocalizedString value = 'AirReservationServices' /> </rim:Name>
</rim:ClassificationNode>

<rim:RegistryPackage id = 'RP-AirServicesANDReservationServices'>
    <rim:Name> <rim:LocalizedString value = 'RP-AirServicesANDReservationServices'/>
    </rim:Name>
</rim:RegistryPackage>

<rim:Association id = 'firstMember' associationType = 'HasMember'
    sourceObject = 'RP-AirServicesANDReservationServices' targetObject = 'AirServices' >
    <rim:Name> <rim:LocalizedString value = 'firstMember' /> </rim:Name>
</rim:Association>

<rim:Association id = 'secondMember' associationType = 'HasMember'
    sourceObject = 'RP-AirServicesANDReservationServices' targetObject = 'ReservationServices' >
    <rim:Name> <rim:LocalizedString value = 'secondMember' /> </rim:Name>
</rim:Association>

<rim:Association id = 'intersectionOf' associationType = 'intersectionOf'
    sourceObject = 'AirReservationServices' targetObject = 'RP-AirServicesANDReservationServices' >
    <rim:Name> <rim:LocalizedString value = 'intersectionOf' /> </rim:Name>
</rim:Association>
```

When such a representation is used to create a complex class in RIM, it becomes possible to infer that the objects classified by both of the classes constituting the intersection are also the instances of this complex class. The following stored procedure retrieves the direct instances of the complex class and also the intersection of the instances of the member classes:

```
CREATE PROCEDURE findInstances($className) AS
BEGIN
SELECT N1.value
FROM Name_ N1, Service S, (
    SELECT A.targetObject AS id
    FROM RegistryPackage R, Association A
    WHERE R.id=A.sourceObject AND
          A.associationType = 'HasMember' AND
          R.id IN (
            SELECT A.targetObject
        FROM Association A, Name_ N, ClassificationNode C
        WHERE A.associationType LIKE 'intersectionOf' AND
              C.id = N.parent AND
              N.value LIKE $className AND
              A.sourceObject = C.id
        )
    ) AS T1, (
    SELECT A.targetObject AS id
    FROM RegistryPackage R, Association A
    WHERE R.id=A.sourceObject AND
          A.associationType = 'HasMember' AND
          R.id IN (
            SELECT A.targetObject
        FROM Association A, Name_ N, ClassificationNode C
        WHERE A.associationType LIKE 'intersectionOf' AND
              C.id = N.parent AND
              N.value LIKE $className AND
              A.sourceObject = C.id
        )
    ) AS T2
WHERE S.id IN (
    SELECT classifiedObject
    FROM Classification
    WHERE classificationNode=T1.id
    INTERSECT
    SELECT classifiedObject
    FROM Classification
    WHERE classificationNode=T2.id
    )  AND T1.id!=T2.id AND
    N1.parent=S.id
UNION
SELECT N.value
FROM Service S, Name_ N
WHERE S.id IN (
    SELECT classifiedObject
    FROM Classification
    WHERE classificationNode IN (
        SELECT id
        FROM ClassificationNode
        WHERE id IN (
            SELECT parent
            FROM name_
            WHERE value LIKE $className
            )
        )
    ) AND S.id=N.parent
END;
```

Table 3 provides a summary of how OWL language elements are mapped to ebXML class hierarchies. In this section, only some of these mappings are explained due to space limitations.

OWL	**ebXML**
owl:Class	ClassificationNode
individual	RegistryObject
rdf:Property	Association
rdfs:domain	sourceObject
rdfs:range	targetObject
owl:equivalentTo owl:samePropertyAs owl:sameAs	An association with a predefined association type of "EquivalentTo".
owl:differentFrom	An new association type of "differentFrom" is defined.
owl:AllDifferent owl:distinctMembers	A new association type "distinctMembers" is defined to add members to a Registry Package.
An association with a new association type is defined.	
rdfs:subClassOf	"subClassOf"
owl:ObjectProperty	"objectProperty"
owl:disjointWith	"disjointWith"
owl:TransitiveProperty	"transitiveProperty"
owl:FunctionalProperty	"functionalProperty"
owl:InverseFunctionalProperty	"inverseFunctionalProperty"
owl:SymetricProperty	"symetricProperty"
owl:DataTypeProperty	XML Schema datatypes are used by providing an external link from the registry.
rdfs:subPropertyOf owl:inverseOf	An association between associations with a new association type "subPropertyOf"/"inverseOf" is defined.
owl:intersectionOf	A registry package is created by associating the classes (i.e. the classification nodes) to be intersected through a new association type of "intersectionOf".
owl:Restriction	Since ebXML RIM associations have local scope, only the type of the Restriction needs to be specified.
A slot type is defined for the association representing a restriction.	
owl:allValuesFrom	"allValuesFrom"
owl:someValuesFrom	"someValuesFrom"
owl:hasValue	"hasValue"
An association with a new association type is defined.	
owl:cardinality	"cardinality"
owl:minCardinality	"minCardinality"
owl:maxCardinality	"maxCardinality"

Table 3. Mapping OWL Ontologies to ebXML Classification Hierarchies without Affecting the Registry

4.2. How to Exploit OWL Lite Semantics for Service Discovery and Composition in ebXML Registry

In this section, in order to demonstrate the benefits of the proposed enhancements to the ebXML registry, we describe a semantic-based service composition tool. This tool partially automates service discovery and composition in OWL aware ebXML registries.

Through a graphical user interface provided, the tool allows the ebXML classification hierarchies to be depicted graphically, as presented in Figure 6. When a user clicks on a node in the classification hierarchy, the generic properties of the service are revealed to the user. The user can fill in the desired properties of services she is looking for through this GUI. The tool queries the ebXML registry automatically to find the services that satisfy user constraints. Then, through a graphical composition tool, the user is allowed to provide choreography of the selected services.

Consider for instance the OWL enriched ontology example given in Figure 7 for the travel domain. Assuming that we want to find all the air reservation services in the registry, it is possible to query the services that are classified under the generic "AirReservationServices" node. In doing this, it is necessary to retrieve the properties of this class so that the user can provide her preferred values for the properties.

Assume that "AirReservationServices" are defined as a subclass of both "OWL-S Profile" class and the "AirServices" class. In conventional ebXML, when a user submits a query to the ebXML registry to get the object properties of the "AirReservationServices", only the immediate associations that are of type "objectProperty" are returned as presented in Figure 8. However by exploiting the semantic capabilities of the OWL aware ebXML, the user can call the stored procedure "findInheritedObjectProperties" defined in Section 4.1.2 to retrieve the properties inherited from the parent classes too (Figure 8).

These properties are shown to the user through the GUI depicted in Figure 9. Once the user provides the preferred values, the instances satisfying these values can be retrieved through the ebXML Filter query shown in Figure 10 which is automatically issued in our tool. Note that while storing the Web service instances, the values of their properties are represented through "slot" values.

Assume further that the AirReservationServices node in the ebXML registry is declared to be equivalent with "OTA_AirReservationServices" through the "EquivalentTo" type association of ebXML.

Without OWL semantic support, when the user issues the Filter Query presented in Figure 10, the ebXML Query Manager will retrieve the services classified only by the AirReservationServices as presented in Figure 11, using the SQL query presented in Figure 12.

With OWL semantic support, our tool processes the semantics of the "EquivalentTo" property to retrieve the instances of the AirReservationServices by using the "findEquivalentInstances(AirReservationServices)" stored procedure defined in

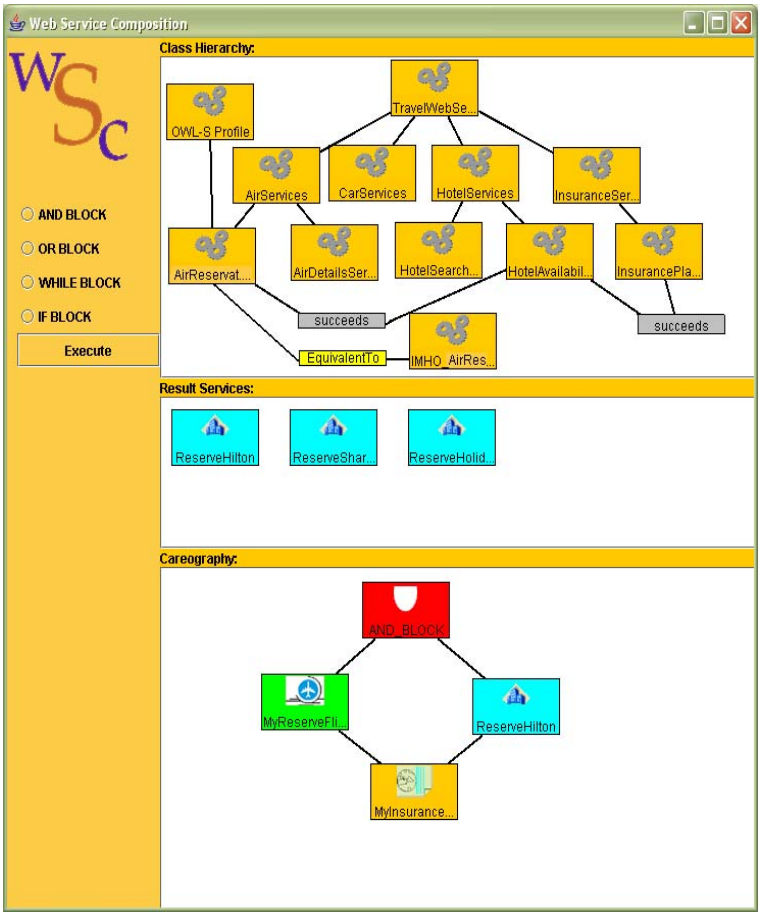

Figure 6. A Snapshot of the GUI tool for semantics-based Web service Composition for ebXML registries

Section 4.1.4. Note that the use of these stored procedures is not restricted to our tool; any ebXML client can also use these stored procedures.

Assuming that the user chooses the "MyAirReservationService" instance among the Web services presented to her, the user may wish to retrieve the "succeeding" services of this instance. Consider the example given in Section 4.1.5. When a user wishes to retrieve the "succeeding" services of the "MyAirReservationService" instance and issues a query to the ebXML registry, without OWL semantic support only "MyHotelAvailabilityService" instance will be returned as presented in Figure 13, although "succeeds" has been declared to be transitive.

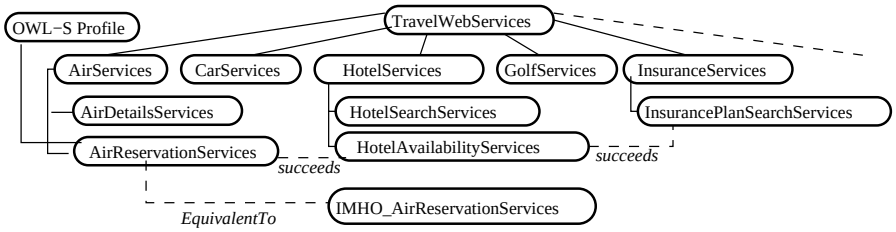

Figure 7. An example Travel Ontology

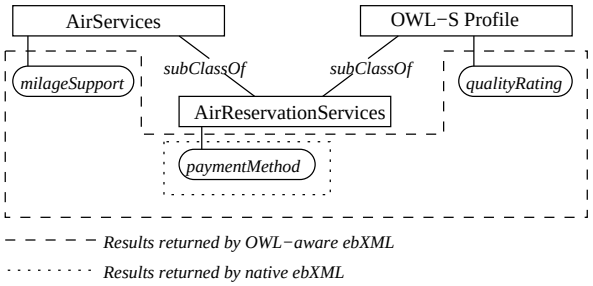

Figure 8. ebXML Semantic support for Class Hierarchies

Figure 9. A GUI tool to obtain the service property values from the user

```
<AdhocQueryRequest >
  <ResponseOption returnType = "LeafClass" returnComposedObjects = "true" />
   <FilterQuery>  <ServiceQuery>
    <ClassifiedByBranch>
        <ClassificationNodeQuery>  <NameBranch>  <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">AirReservationServices </StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationNodeQuery>
      </ClassifiedByBranch>
    <SlotBranch>  <SlotFilter> <Clause>  <SimpleClause leftArgument = "name_">
        <StringClause stringPredicate = "Equal">paymentMethod</StringClause>
                </SimpleClause> </Clause> </SlotFilter>
        <SlotValueFilter> <Clause> <SimpleClause leftArgument = "value">
        <StringClause stringPredicate = "Contains">CreditCard</StringClause>
                </SimpleClause> </Clause> </SlotValueFilter>
    </SlotBranch>
    ...
    </ServiceQuery>   </FilterQuery> </AdhocQueryRequest>
```

Figure 10. An Example Filter Query for Retrieving the Instances of the "AirReservationService"

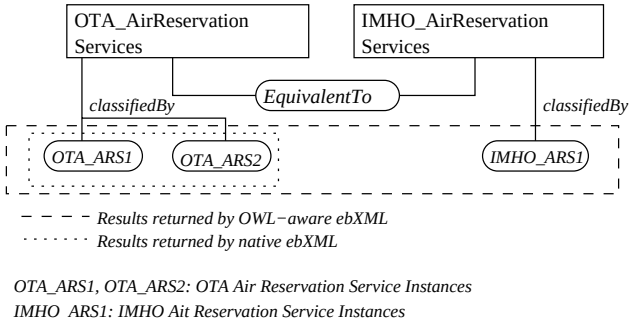

Figure 11. ebXML Semantic support for Equivalent Classes

To be able to exploit the "transitivity" semantics, the user can use the "findTransitiveRelationships(AirReservationServices,succeeds)" stored procedure defined in Section 4.1.5, which will return the "MyInsuranceService" instance additionally.

5. ebXML Semantic Standardization Work

In this section, we briefly summarize the ebXML semantic standardization efforts, in order to put the work described in this paper into perspective. This paper addresses making the ebXML registry OWL aware. There are several other key semantic requirements being addressed within the OASIS open source standards body [26].

```
SELECT *  FROM Service WHERE id IN (
    SELECT classifiedObject FROM Classification
    WHERE classificationNode IN (
        SELECT id FROM ClassificationNode
        WHERE id IN (
            SELECT parent FROM name_
            WHERE value='AirReservationServices'
        )
    )
) AND id IN (
    SELECT parent FROM Slot
    WHERE name_='paymentMethod' AND
    value LIKE '%CreditCard%')
```

Figure 12. SQL query to retrieve the services classifed with "AirReservationServices"

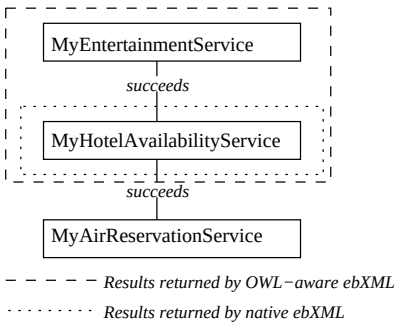

Figure 13. ebXML Semantic support for Transitive Properties

This work is progressing through the committees including: The Business-Centric Methodology (BCM) Technical Committee (TC) [27], The ebXML Registry Semantic Content Management Sub-Committee (ebXMLR-SCM) [28], The UDDI Technical Committee [30] and the Topic Maps Published Subjects Technical Committee [29]. In this section, we focus on the most relevant work by BCM and ebXMLR-SCM:

- The Business-Centric Methodology (BCM) Technical Committee (TC) [27]: BCM addresses a proper interpretation of the business language semantics found in a SOA (Service Oriented Architecture) metadata framework/classification system which is essential for harnessing tacit knowledge and facilitating shared communications. Particularly, the BCM identifies a Conceptual Layer that enables the exploitation of community-of-interest specific classifications, e-business taxonomies and systemic patterns as key factors in semantic interoperability.

- The ebXMLRegistry Semantic Content Management Sub-Committee (ebXMLR-SCM) [28]: A key factor of the ebXMLR-SCM work towards semantic exten-

sions of the Registry/Repository is the acknowledgment that the mapping of e-business artifacts to a semantic structure can employ many types of registry objects. For instance, a simple lexicon containing the definition of words used by a particular group of professionals may be formatted as a data dictionary that is referenced by other objects, such as, UN/CEFACT Core Components [44]. Thesauri classification objects, employing terms of a controlled vocabulary which are associated via parent-child relationships, are most likely to be used for the indexing of community-of-interest specific information. Taxonomies supporting the systemic cataloging of e-business components should be capable of categorizing a mix of Nouns/Verbs, information-based structures (e.g. XML Schemas), behavior-based definitions (e.g. WSDL) and process-based specifications (e.g. BPSS). Indeed, the ebXMLR-SCM recognized that, since these and other semantic structures co-exist in the real-world, the Semantic Content Management supporting a federation of Registries/Repositories must also allow them to co-exist.

6. Related Work

In the early nineties, ontologies have been a research topic being addressed in a rather small research community. This changed drastically in the late nineties by the insight that a conceptual, yet executable model of an application domain provides a significant value [17, 42]. The impact has increased with the Semantic Web initiative and the Web Ontology Language (OWL) [31].

The importance of semantics is also recognized in the Web services area and there have been several efforts to improve the semantics support for Web services. An important effort is OWL-S [32] which defined an upper ontology to describe service semantics.

The need for extending the UDDI [43] registries with semantic capabilities has been addressed in the literature [9, 10, 34]. Note that UDDI registries use tModels to represent compliance with a taxonomy such as Universal Standard Products and Services Classification [45]. [9, 10] describe a mechanism to relate DAML-S ontologies with services advertised in the UDDI registries. [34] also addresses importing semantic to UDDI registries where DAML-S specific attributes such as *inputs, outputs* and *geographicRadius* are represented using tModel mechanisms of UDDI. The matching engine implemented is based on the algorithm described in [33].

An extended UDDI registry is reported in [39] which allows to record user defined properties associated with a service and then to enable discovery of services based on these.

In [40], the authors discuss adding semantics to WSDL using DAML+OIL ontologies. Their approach also uses UDDI to store these semantic annotations and search for Web services based on them.

[23] discusses the applicability of the DAML-S profile, process model, and grounding ontologies to the Web service lifecycle.

In [47], DAML-S is extended to describe bioinformatics Web services and the services are matched by subsumption reasoning over the service descriptions.

Since the semantic support provided by UDDI and ebXML registries differ considerably, it is not possible to repeat the previous work in UDDI for ebXML.

Related with ebXML, exploiting the class hierarchies in ebXML registries for service discovery and composition is described in [11]. In [12], some initial ideas about enriching ebXML registries with OWL semantics is presented.

In [4], a conceptual architecture called Web Service Modeling Framework (WSMF) is described based on the principles of strong decoupling and mediation of services.

A tutorial on semantic of Web services is available at [3] where a detailed overview of Web Service Modeling Framework (WSMF) is given. The WSMF consists of four main elements:

- Ontologies that provide the terminology used by other elements,
- Goal repositories that define the problems that should be solved by web services,
- Web services descriptions that define various aspects of a web service, and
- Mediators which bypass interoperability problems.

[5] describes an algorithm to discover Web services and resolve heterogeneity among their interfaces and the workflow host.

[24] proposes an ontology-based framework for the automatic composition of Web services. The authors present a technique to generate composite services from high-level declarative descriptions. For this purpose, they extend WSDL with semantic capabilities.

An insightful description of Web services is given in [25] where the authors put the Web service technologies into perspective in Business-to-Business interaction domain.

7. Conclusions and Future Work

This paper describes an engineering effort on how an ebXML registry can be made OWL aware. The work presented provides the foundation for OWL representations to be expressed in the registry. The representation of OWL semantics directly in the RIM enables the standard ebXML query facility to use stored procedures that can return a richer set of results based on explicit OWL constructs. As demonstrated, the queries of this type provide new capabilities to the ebXML client applications.

In this work, we use OWL Lite, since we are using ontologies to get knowledge through querying rather than reasoning. We investigate the possible ways of making the registry OWL aware and describe an approach that minimizes the changes on the ebXML specification.

There are two observations resulting from this experience:

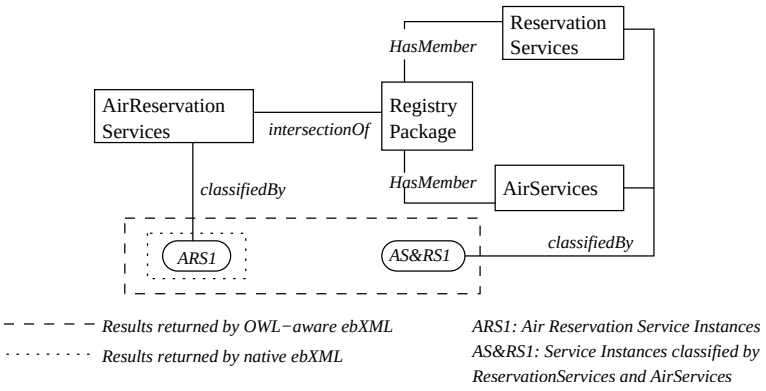

Figure 14. ebXML Semantic support for Class Intersection

- Ontologies can play two major roles: one is to provide a source of shared and precisely defined terms which can be used formalizing knowledge and relationship among objects in a domain of interest. The other is to reason about the ontologies. When an ontology language like OWL is mapped to a class hierarchy like the one in ebXML, the first role can directly be achieved. However, when we want to infer new information from the existing knowledge, we need reasoners. And reasoners can not directly run on the ebXML registry because all the registry information is stored in relational databases. Hence, there is a need to reconstruct the ontology from its representation in the ebXML registry.

- An ebXML registry client can use stored procedures that we have introduced to handle the OWL semantics. However, handling this semantics through the filter query in a transparent way to the user requires some modifications in the Query Manager Component of the registry. ebXML filter query, is designed to retrieve the registry objects as specified in the original RIM. It falls short to retrieve additional semantics introduced in this work.

  In "filter query", the user expresses what is to be retrieved from the registry as an XML message and the current syntax of ebXML query uses the conventional ebXML registry constructs. In order to retrieve extended semantics from in an OWL aware ebXML registry, through a "filter query", the Query Manager component needs to be extended.

  Consider the example defined in Section 4.1.8, where "AirReservationServices" is defined to be the intersection of the classes "AirServices" and "ReservationServices". When a user sends a Filter query to retrieve services classified by the "AirReservationServices" node, normally the ebXML Query Manager will return the services directly classified by "AirReservationServices" node. However with OWL support it is possible to retrieve the services classified by both of the

"AirServices" and "ReservationServices" at the same time, and thus retrieving the instances of "AirReservationServices", as presented in Figure 14.

To handle such a semantics, the ebXML Query Manager should be updated to execute the "findInstances($className) stored procedure defined in Section 4.1.8 whenever it receives such a filter query. In fact, the Query Manager needs to consider all such possibilities and this can only be handled through reasoning.

There are a number of public domain and commercial OWL reasoners such as [6, 13, 36]. As a future work, we intend to improve the Query Manager component with reasoning capabilities by exploiting one of the existing OWL reasoners.

References

1. Antoniou, G., Harmalen, F., "Web Ontology Language: OWL", in Handbook on Ontologies, Springer, 2004.
2. Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, May 2001.
3. Bussler, C., Fensel, D., Payne, T., Sycara, K., "ISWC Tutorial: Semantic Web Services", http://www.daml.ri.cmu.edu/tutorial/iswc-t3.html#2b
4. Bussler, C., Fensel, D., Maedche, A., "A Conceptual Architecture for Semantic Web Enabled Web Services", ACM Sigmod Record, Vol. 31, No. 4, December 2002.
5. Cardoso, J., Sheth, A., "Semantic e-Workflow Composition", Journal of Intelligent Information Systems (JIIS), Vol. 12, No. 3, 2003.
6. Cerebra OWL Reasoner, http://www.networkinference.com/Products/Cerebra_Server.html
7. DAML+OIL Reference Description, W3C Note, http://www.w3.org/2001/10/daml+oil, December 2001.
8. DAML Services Coalition (A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web Services", in Proceedings of the International Semantic Web Working Symposium (SWWS), July 2001.
9. Dogac, A., Cingil, I., Laleci, G. B., Kabak, Y., "Improving the Functionality of UDDI Registries through Web Service Semantics", 3rd VLDB Workshop on Technologies for E-Services (TES-02), Hong Kong, China, August 23-24, 2002.
10. Dogac, A., Laleci, G., Kabak, Y., Cingil, I., "Exploiting Web Service Semantics: Taxonomies vs. Ontologies", IEEE Data Engineering Bulletin, Vol. 25, No. 4, December 2002.
11. Dogac, A., Kabak, Y., Laleci, G., "A Semantic-Based Web Service Composition Facility for ebXML Registries", 9th International Conference of Concurrent Enterprising, Espoo, Finland, June 2003.
12. Dogac, A., Kabak, Y., Laleci, G., "Enriching ebXML Registries with OWL Ontologies for Efficient Service Discovery", in Proc. of RIDE'04, Boston, March 2004.
13. F-OWL Reasoner, http://fowl.sourceforge.net
14. ebXML, http://www.ebxml.org/
15. ebXML Registry Information Model v2.5, http://www.oasis-open.org/committees/regrep/-documents/2.5/specs/ebRIM.pdf
16. ebXML Registry Services Specification v2.5, http://www.oasis-open.org/committees/-regrep/documents/2.5/specs/ebRIM.pdf
17. Fensel, D., Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Springer, 2001.
18. freebXML Registry Open Source Project http://ebxmlrr.sourceforge.net
19. Horrocks, I., "DAML+OIL: A Description Logic for the Semantic Web", IEEE Data Engineering Bulletin, Vol. 25, No. 1, March 2002.

20. IST-2104-SATINE Project, http://www.srdc.metu.edu.tr/webpage/projects/satine/
21. Jena2 Semantic Web Toolkit, http://www.hpl.hp.com/semweb/jena2.htm
22. McGuinness, D., Harmelen, F., "OWL Web Ontology Language Overview", W3C Recommendation, February 2004, http://www.w3.org/TR/owl-features/
23. McIlraith, S. A., Martin, D. L., "Bringing Semantics to Web Services", IEEE Intelligent Systems, Vol.18, No.1, 2003.
24. Medjahed, B., Bouguettaya, A., Elmagarmid, A., "Composing Web services on the Semantic Web", VLDB Journal, Vol.12, No.4, 2003.
25. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A., Elmagarmid, A., "Business-to-business interactions: issues and enabling technologies", VLDB Journal, Vol.12, No.1, 2003.
26. OASIS, http://www.oasis-open.org/home/index.php
27. OASIS Business-Centric Methodology (BCM) Technical Committee (TC), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=bcm
28. OASIS ebXML Registry Semantic Content Management SC, http://www.oasis-open.org/apps/org/workgroup/regrep-semantic/
29. OASIS Topic Maps Published Subjects TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tm-pubsubj
30. OASIS UDDI Specification TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec
31. OWL Web Ontology Language 1.0 Reference http://www.w3.org/TR/2002/WD-owl-ref-20020729/ref-daml
32. OWL-S, http://www.daml.org/services/daml-s/0.9/
33. Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "Semantic Matching of Web Services Capabilities", in Proc. of Intl. Semantic Web Conference, Sardinia, Italy, June 2002.
34. Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "Importing the Semantic Web in UDDI", in Web Services, E-Business and Semantic Web Workshop, 2002.
35. Registering web services in an ebXML Registry version 1.0. http://www.oasis-open.org/apps/org/workgroup/regrep/download.php/1636/OASIS-Registry
36. Pellet OWL Reasoner, http://www.mindswap.org/2003/pellet/
37. RDF Schema: Resource Description Framework Schema Specification, W3C Proposed Recommendation, 1999, http://www.w3.org/TR/PR-rdf-schema.
38. RDF Syntax: Resource Description Framework Model and Syntax Specification, W3C Recommendation, 1999, http://www.w3.org/TR/REC-rdf-syntax.
39. ShaikhAli, A., Rana, O., Al-Ali, R., Walker, D., "UDDIe: An Extended Registry for Web Services", Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference, Florida, January 2003.
40. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., "Adding Semantics to Web Services Standards", In proc. of ICWS, 2003.
41. Smith, M., Welty, C., McGuinnes, D., "OWL Web Ontology Language Guide", W3C Recommendation, February 2004, http://www.w3.org/TR/owl-guide/
42. Staab, S., Studer, R., Handbook on Ontologies, Springer, 2004.
43. Universal Description, Discovery and Integration (UDDI), www.uddi.org
44. UN/CEFACT-ebXML
Core Components Technical Specification, www.oasis-open.org/committees/download.php/4259/CEFACT%20CCTS%20Version%202%20of%2011%20August.pdf
45. Universal Standard Products and Services Classification (UNSPSC) http://eccma.org/unspsc
46. Web Content Management using ebXML Registry http://ebxmlrr.sourceforge.net/presentations/xmlEurope2004/04-02-02.pdf
47. Wroe, C., Stevens, R., Goble, C., Roberts. A., Greenwood, M., "A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data", Intl. Journal of Cooperative Information Systems, to appear.